

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO ĐỒ ÁN 1
COLOR COMPRESSION

GV hướng dẫn	: Nguyễn Văn Quang Huy
	: Nguyễn Ngọc Toàn
Sinh viên thực hiện	: Lê Thị Hồng Hạnh
MSSV	: 22127103
Lớp	: 22CLC07

MỤC LỤC

1. Ý tưởng thực hiện	2
2. Mô tả các hàm	3
2.1. Hàm đọc ảnh.....	3
2.2. Hàm hiển thị ảnh	3
2.3. Hàm lưu ảnh	3
2.4. Hàm chuyển <i>img_2d</i> sang <i>img_1d</i>	3
2.5. Hàm K-Means.....	3
2.6. Hàm tạo ảnh mới	4
2.7. Hàm khởi tạo <i>centroids</i> (hàm cài thêm).....	5
2.8. Hàm tạo file path ảnh mới với extension yêu cầu (hàm cài thêm)	5
2.9. Hàm main.....	5
3. Chương trình demo.....	6
4. Nhận xét.....	10
Tài liệu tham khảo	11

1. Ý tưởng thực hiện

K-Means Clustering là một thuật toán máy học không giám sát (nghĩa là dữ liệu không có nhãn) dùng để giải quyết bài toán phân cụm dữ liệu. Mục tiêu của thuật toán là phân cụm một tập dữ liệu lớn thành k cụm dữ liệu với centroid (tâm cụm) là điểm đặc trưng cho một cụm có giá trị bằng trung bình của toàn bộ dữ liệu trong cụm. Dựa vào khoảng cách từ centroid đến các điểm dữ liệu để xác định label (nhãn) cho chúng thuộc về centroid gần nhất. Sau khi hoàn thành việc xác định nhãn cho tất cả dữ liệu, centroid sẽ được cập nhật lại để đảm bảo centroid nằm giữa cụm. Việc xác định label cho các điểm dữ liệu và cập nhật centroid sẽ được thực hiện lặp lại cho đến khi vị trí centroid không thay đổi.

Các bước thực hiện cụ thể đối với bài toán Color Compression

- Đọc dữ liệu ảnh từ đường dẫn ảnh và chuyển về dạng matrix, lưu ý đảm bảo là ảnh RGB trước khi chuyển về matrix. Mỗi dòng của matrix có 3 channel và đại diện cho 1 pixel.
- Áp dụng K-Means để phân cụm các pixel trong ảnh thành k cụm:
 - Bước 1: Khởi tạo k centroid.
 - Bước 2: Tính khoảng cách của mỗi pixel đến k centroids để xác định centroid nào gần nó nhất và gán giá trị cluster label cho từng điểm ảnh.
 - Bước 3: Cập nhật lại giá trị centroid.
 - Bước 4: Lặp lại bước 2, 3 cho đến khi gặp lặp đủ `max_iter` hoặc vị trí của centroid không thay đổi nữa.
- Từ labels ta tiến hành gán giá trị của centroids cho từng điểm ảnh để tạo ra ảnh mới.
- Lưu ảnh mới đã được nén với extension được yêu cầu vào đường dẫn.

2. Mô tả các hàm

2.1. Hàm đọc ảnh

```
def read_img(img_path)
```

Từ thư viện PIL, ta sử dụng hàm `open()` của module `Image` để đọc ảnh từ đường dẫn và trả về giá trị có type là class ***PIL.Image***. Ta chuyển về class ***numpy.ndarray*** bằng hàm `array()` của thư viện `numpy` để dễ dàng xử lý, lưu ý đảm bảo chuyển về ảnh RGB trước khi chuyển về matrix.

Note:** Do thư viện PIL được sử dụng cho đọc và lưu ảnh nên việc chuyển đổi giữa ***PIL.Image và ***numpy.ndarray*** được diễn ra trong 2 hàm đọc và lưu ảnh. Trong các hàm khác thì biến *img* mặc định có type là ***numpy.ndarray***.*

2.2. Hàm hiển thị ảnh

```
def show_img(img_2d)
```

Từ thư viện `matplotlib`, ta sử dụng hàm `imshow()` của module `pyplot` để hiển thị ảnh. Đối số truyền vào có thể là `PIL.Image` hoặc `numpy.ndarray`

2.3. Hàm lưu ảnh

```
def save_img(img_2d, img_path)
```

Từ thư viện PIL, ta sử dụng hàm `save()` của module `Image` để lưu ảnh vào đường dẫn và extension được yêu cầu. Ta cần chuyển biến `img_2d` từ ***numpy.ndarray*** thành ***PIL.Image*** bằng hàm `fromarray()` của module `Image` và xác định extension của file ảnh từ đường dẫn file ảnh trước khi sử dụng hàm `save()`. Hàm sẽ raise `KeyError` và `IOError` có thể xảy ra khi thực hiện `save()`.

2.4. Hàm chuyển `img_2d` sang `img_1d`

```
def convert_img_to_1d(img_2d)
```

Từ thư viện `numpy`, ta sử dụng hàm `reshape()` để chuyển đổi matrix kích thước 2D (height, width, channels) sang 1D (height*width, channels). Do data type của các phần tử trong matrix là ***numpy.uint8*** nên ta phải chuyển về ***float*** để tránh sai sót khi tính toán khoảng cách giữa các pixels và *centroids*.

2.5. Hàm K-Means

```
def kmeans(img_1d, k_clusters, max_iter, init_centroids)
```

Ta thực hiện phân cụm màu theo các bước của ý tưởng thực hiện đã nêu ở trên. Khởi tạo k *centroids* sẽ được thực hiện bởi hàm ***initialize_centroids()*** được cài đặt thêm (mô tả hàm sẽ được nêu ở phần sau), data type của các phần tử *centroids* cũng là ***float*** để dễ dàng cho việc tính toán.

Tính toán khoảng cách từ mỗi pixels đến mỗi *centroids* và kết quả được lưu vào biến *distance*. Tức ta phải lấy từng phần tử của mảng *img_1d* trừ cho từng phần tử của *centroids* rồi lấy chuẩn của chúng nhưng shape của *img_1d* là (height*width, channels) và *centroids* là (k, channels) nên không thể thực hiện trừ. Nhận thấy *difference* – matrix hiệu của *img_1d* và *centroids* mong muốn nhận được có shape (k, height*width, channels), ta có thể sử dụng ***Broadcasting***. Ta thêm một chiều vào *centroids* bằng ***numpy.newaxis*** để có được shape mới là (k, 1, channels) và *img_1d* (height*width, channels). Lúc này, có thể thực hiện *img_1d* – *centroids* vì *centroids* có một chiều bằng 1, khi đó chiều này sẽ dẫn giãn ra (height*width) để tương ứng với shape của *img_1d* (theo quy tắc của ***Broadcasting***) nên ta sẽ nhận được *difference* có shape(k, height*width, channels). Sử dụng hàm ***numpy.linalg.norm()*** để tính chuẩn của *difference* và lưu vào *distance* có shape (k, height*width).

Sử dụng hàm ***numpy.argmin()*** với đối số *distance* và axis=0 (lấy phần tử nhỏ nhất ở mỗi cột của matrix *distance*) để lấy được index của centroid mà pixels đó gần nhất lưu vào *labels*.

Cập nhật giá trị của k *centroids* bằng cách tính trung bình tất cả các điểm ảnh thuộc cluster đó. Với centroid thứ j , ta tìm tất cả phần tử của *img_1d* thuộc cluster thứ j bằng (*labels* của phần tử đó bằng j). Tính trung bình các giá trị trong cluster j bằng ***numpy.average()*** và đó cũng là giá trị centroid mới *new_centroids* thứ j .

Kiểm tra các phần tử của *centroids* và *new_centroids* có bằng nhau (element-wise) với một sai số (trong đồ án sai số là 1). Nếu True thì kết thúc vòng lặp, False thì *centroids* được cập nhật bằng *new_centroids* và lặp lại các bước tính khoảng cách, gán *labels*, cập nhật *centroids* đến khi chạy hết *max_iter* hoặc *centroids* và *new_centroids* bằng nhau với sai số 1.

Khi kết thúc vòng lặp, hàm trả về *centroids* và *labels*.

2.6. Hàm tạo ảnh mới

```
def generate_2d_img(img_2d_shape, centroids, labels)
```

Tạo matrix mới với giá trị các pixels là giá trị *centroids* của chúng bằng List Comprehension. Chuyển matrix ảnh mới về shape ban đầu của matrix ảnh đầu vào và data type ***numpy.uint8***.

2.7. Hàm khởi tạo *centroids* (hàm cài thêm)

```
def initialize_centroids(img_1d, k_clusters, init_centroids)
```

Sử dụng hàm ***numpy.random.choice()*** với *replace=False* để generate matrix *centroids* tùy theo method được yêu cầu và không trùng lặp giá trị. Có 2 method khởi tạo:

‘random’: *centroids* được khởi tạo với mỗi channels là một giá trị bất kì trong [0, 255]. Đối số được truyền vào ***numpy.random.choice()*** là 256 (generate giá trị từ 0 đến 255) và shape của *centroids*.

‘in_pixels’: một pixels từ ảnh gốc được chọn làm *centroids*. Ta chỉ việc generate k index của *img_1d* và gán giá trị *img_1d[index]* cho *centroids*, nhưng sẽ xảy ra sự trùng lặp *centroids* vì có thể xảy ra *img_1d[i] = img_1d[j]* với $i \neq j$. Do đó ta tách các phần tử duy nhất của *img_1d* thành *unique_rows* bằng hàm ***numpy.unique()***. Sau đó generate k index của *unique_rows*.

2.8. Hàm tạo file path ảnh mới với extension yêu cầu (hàm cài thêm)

```
def create_new_img_path(old_img_path, extension)
```

Từ file path đầu vào, ta tìm được index của dấu ‘.’ từ cuối string bằng hàm ***rfind()***. File path mới được ghép từ file path cũ thêm ‘_compressed’ và extension muốn lưu.

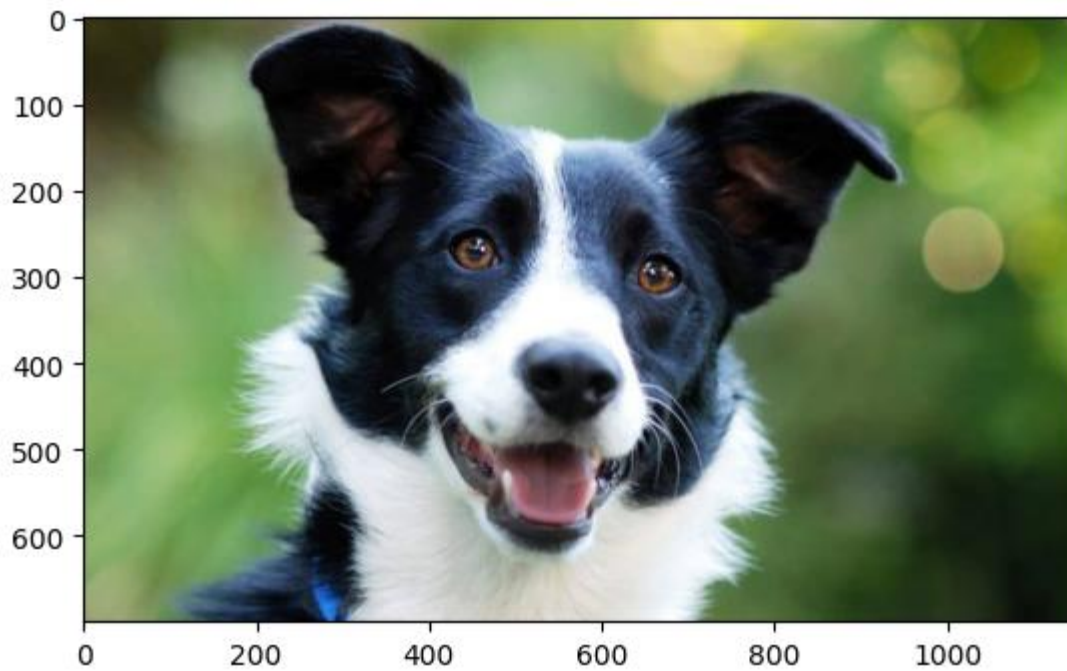
2.9. Hàm main

```
def main()
```

- Nhập file path
- Đọc ảnh
- Chuyển từ ảnh 2D (height, width, channels) sang 1D (height*width, channels)
- Thực hiện phân cụm bằng K-Means
- Tạo ảnh mới từ *centroids* và *labels* được trả về từ K-Means
- Hiển thị ảnh sau khi xử lý
- Lưu ảnh

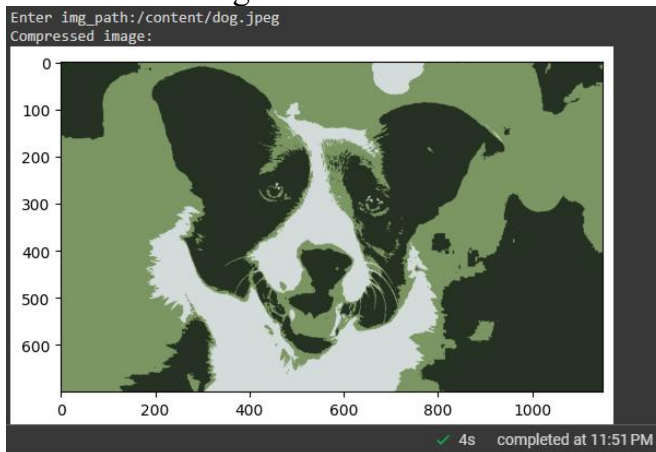
3. Chương trình demo

Input: dog.jpeg

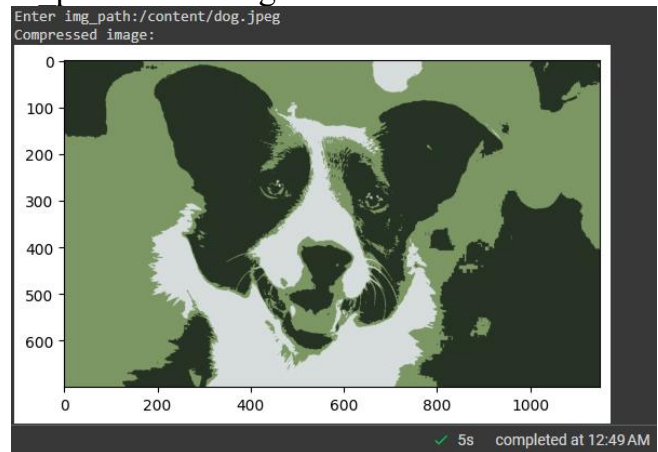


Với $k = 3$

random: excuting time = 4s



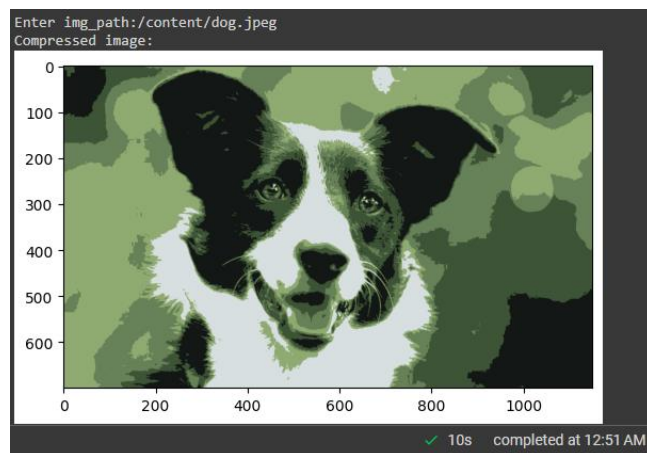
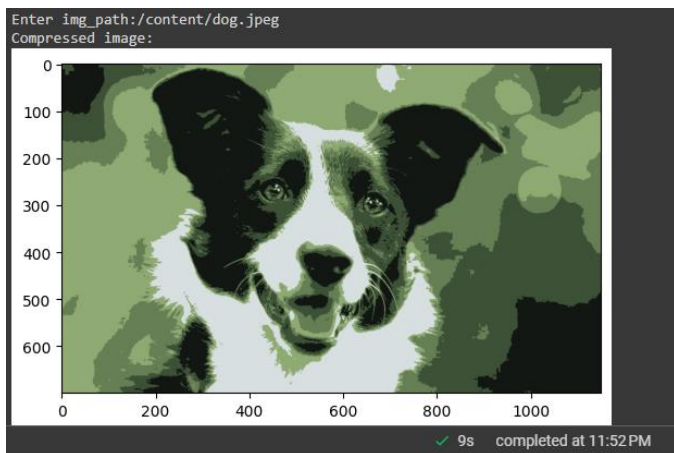
in_pixels: excuting time = 5s



Với $k = 5$

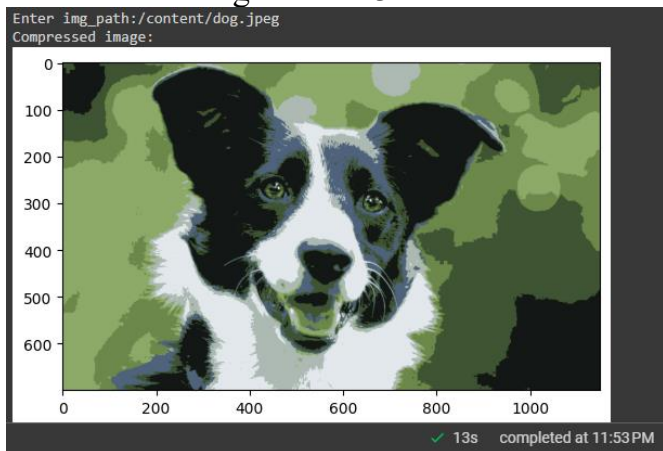
random: excuting time = 9s

in_pixels: excuting time = 10s

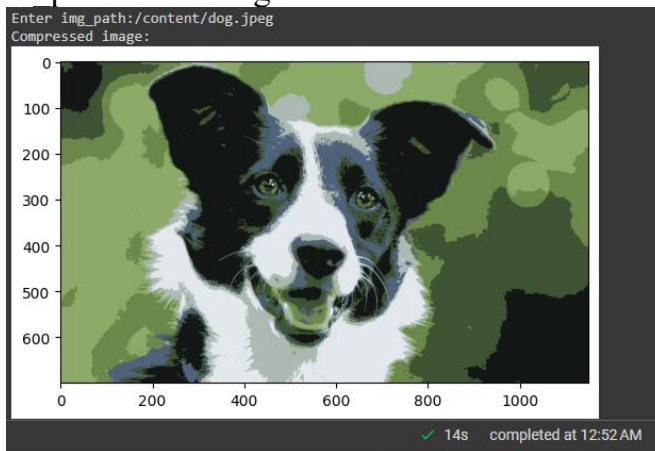


Với $k = 7$

random: excuting time = 13s

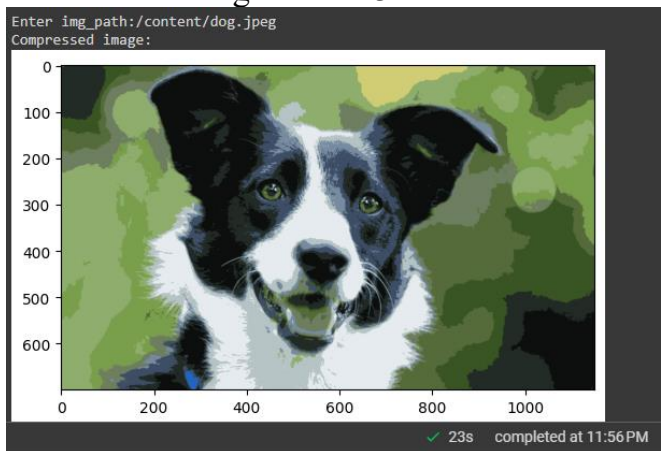


in pixels: excuting time = 14s

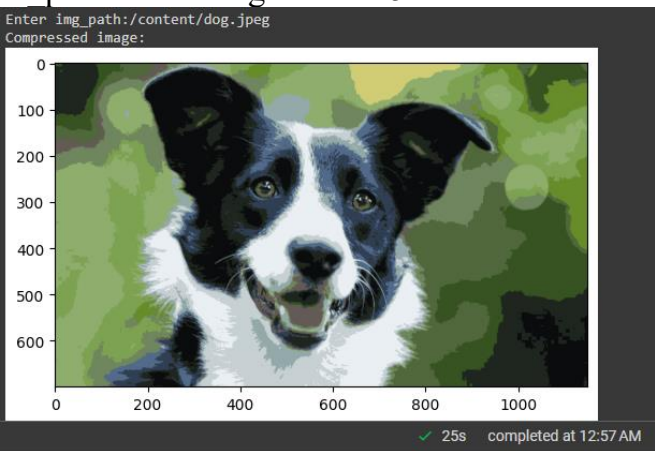


Với $k = 15$

random: excuting time = 23s

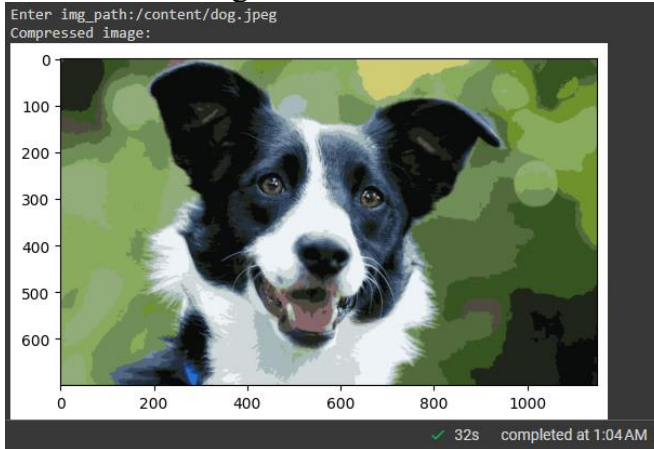


in pixels: excuting time = 25s

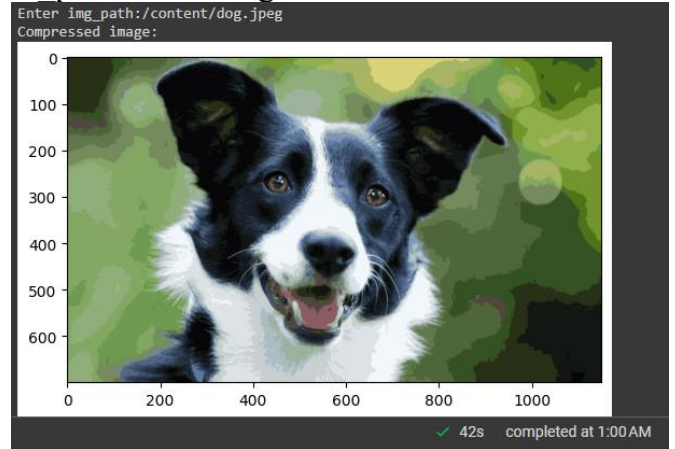


Với $k = 30$

random: excuting time = 32s

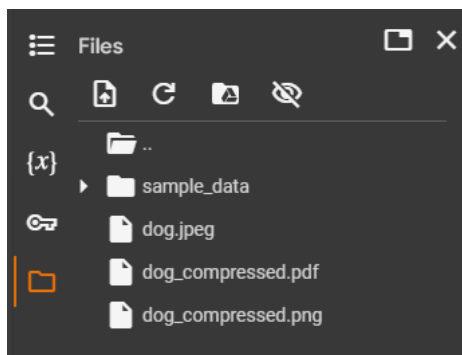


in_pixels: excuting time = 42s

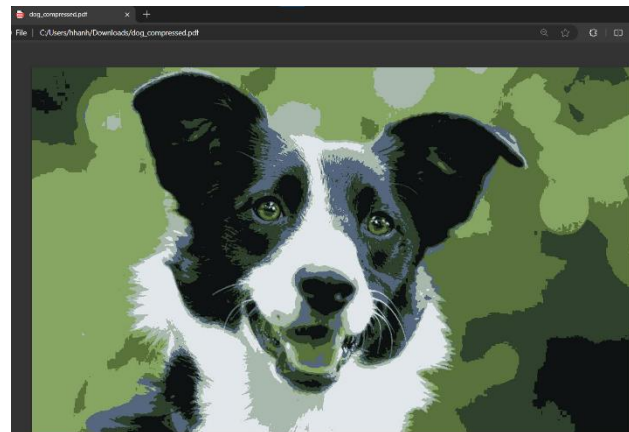
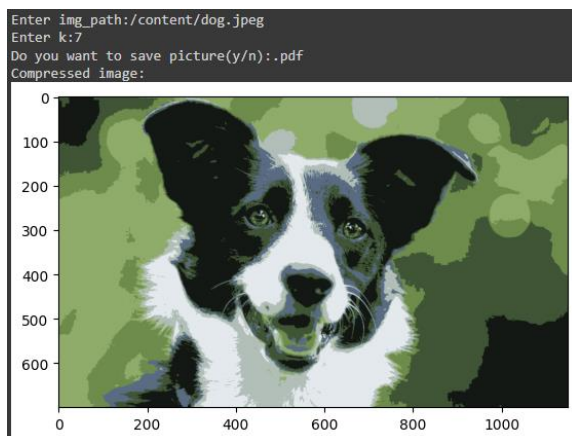


Ngoài ra, chương trình cho phép lưu ảnh nếu người dùng muốn với file path là file path ảnh gốc + ‘_compressed’ + extension. Ví dụ muốn lưu ảnh với extension là .pdf cho output này, file path ảnh mới là ../dog_compressed.pdf. Lưu ý, nếu extension không hợp lệ hoặc không thể lưu ảnh với extension đó thì hàm chương trình sẽ raise error.

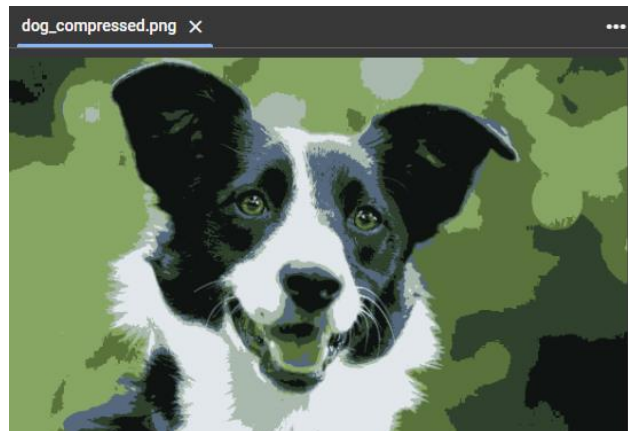
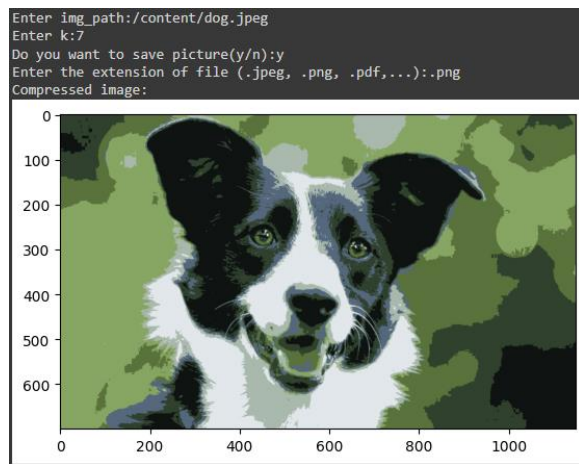
Lưu ảnh với extension .pdf và .png cho ra kết quả:



Ảnh .pdf






Ảnh .png



4. Nhận xét

Như ta thấy ở chương trình demo trên, k càng lớn thì hình ảnh cho ra càng chi tiết nhưng đồng thời thời gian thực thi cũng lâu hơn do phải generate nhiều centroid hơn, tính toán khoảng cách với nhiều centroid hơn và phải cập nhật nhiều centroid hơn,... Giữa khởi tạo *centroids* 'random' hay 'in_pixels' không có quá nhiều sự khác nhau ở thời gian thực thi, nhưng ta có thể thấy 'in_pixels' có những cụm màu sắc giống ảnh gốc hơn 'random' do *centroids* được generate từ các pixels trong ảnh gốc. Ngoài ra, `max_iter` cũng ảnh hưởng đến thuật toán vì nếu `max_iter` quá nhỏ thì số lần cập nhật centroids và labels sẽ không đủ và ảnh cho ra sẽ không như mong đợi, trong đồ án `max_iter=100` cho ra kết quả khá ổn định. Đồng thời khi k càng lớn, ảnh cho ra càng có nhiều màu nên dung lượng ảnh cũng sẽ cao hơn.

 dog_compressed (2).pdf	=> $k = 7$	6/16/2024 11:42 PM	Microsoft Edge PD...	68 KB
 dog_compressed (1).pdf	=> $k = 5$	6/16/2024 11:41 PM	Microsoft Edge PD...	65 KB
 dog_compressed.pdf	=> $k = 3$	6/16/2024 11:41 PM	Microsoft Edge PD...	57 KB

Nhìn chung các kết quả cho ra đúng như mong đợi, giá trị của k , centroids và `max_iter` ảnh hưởng nhiều đến performance của thuật toán. Tùy vào input ta phải ước lượng giá trị của các biến trên cho phù hợp để tối ưu và cho ra kết quả hiệu quả nhất.

Tài liệu tham khảo

Numpy Documentation: <https://numpy.org/doc/>

PIL:

<https://viblo.asia/p/huong-dan-su-dung-thu-vien-pillow-de-xu-ly-hinh-anh-trong-python-cho-nguoi-moi-bat-dau-3Q75wm4MZWb>

<https://www.geeksforgeeks.org/python-pil-image-open-method/>

<https://www.geeksforgeeks.org/python-pil-image-save-method/>

matplotlib.pyplot: <https://matplotlib.org/stable/tutorials/images.html>

AI: <https://gemini.google.com/>

Stackoverflow: <https://stackoverflow.com/questions/>

<https://stackoverflow.com/questions/10965417/how-to-convert-a-numpy-array-to-pil-image-applying-matplotlib-colormap>

<https://stackoverflow.com/questions/384759/how-do-i-convert-a-pil-image-into-a-numpy-array>

<https://stackoverflow.com/questions/16970982/find-unique-rows-in-numpy-array>